**DEFINICION DEL ESQUEMA, RESTRICCIONES, CONSULTAS Y VISTAS**

**ESQUEMA**

El concepto es de esquema es utilizado para el agrupamiento de tablas y otras estructuras pertenecientes a la misma aplicación de la base de datos.

|  |
| --- |
| **CATALOGO** |

-INFORMATION\_SCHEMA

|  |  |  |
| --- | --- | --- |
| **ESQUEMA** | | |
| *Nombre de esquema* | *Identificador de autorización* | *Descriptores por elemento* |

*Elementos=tablas, restricciones, vistas, dominios, otras estructuras como (concesión de autorización). Descriptores del esquema.*

* *Ejemplo*

*La siguiente sentencia crea el esquema EMPRESA, propiedad del usuario cuyo identificador de autorización es jperez.*

**Create schema** empresa authorization jperez;

\*No todos los usuarios están autorizados a crear esquemas y elementos del esquema. El único es nombrado por el administrado de la base datos.

**Tipos de datos:**

* Numéricos: INTEGER, INT, SMALLINT, FLOAT, DEC, NUMERIC.
* Cadena de caracteres: CHAR, CHARACTER, VARCHAR, CHAR VARING, CHARACER VARING.
* Cadena de bits: BIT, BIT BINARING.
* Booleano: TRE, FALSE.
* Fecha y hora: DATE(año, mes , día) (AAAA-MM-DD), (horas, minutos, segundos) (HH:MM:SS)
* Marca de tiempo: TIMESTAMP ‘2002-09-27 09:12:47’.

**Create domain** tipo\_dni as char (9)

**RESTRICCIONES**

Herramientas para la creación de tablas donde los datos a ingresar tengan que ser validados por medio de valores numéricos.

* Retricciones de clave e integridad referencial

**PRIMARY KEY:** Especifica uno o más atributos que constituyen a clave principal de una relación.

**FOREING KEY:** Identifica la integridad referencial.

* dominios de atributo

**NOT NULL:** Implícitamente en los atributos // cuando es necesario un valor, es decir

El campo no puede estar vacio.

**DEFAULT:** Predeterminado si no se hace la utilización del NOT NULL.

* tuplas individuales dentro de una relación

**CHECK:** Restringe valores dentro de un rango o utilizando operadores de comparación.

* Asignación de nombres a las restricciones

**CONSTRAINT:** Nombre de restricción se utiliza para identificar una restricción en particular en caso de que la restricción tenga que eliminarse mas tarde.

**CONSULTAS**

* Estructura SELECT-FROM-WHERE de las consultas básicas de SQL

**SELECT:** < Lista de atributos>

**FROM:** <lista de tablas>

**WHERE**:<codición>

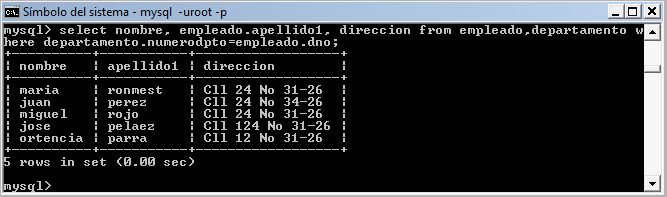
* Nombre de atributos ambiguos, alias y variables de tupla

Si el mismo nombre se puede utilizar para dos (o más) atributos, siempre y cuando los atributos se encuentren en relaciones diferentes. Si es el caso, y una consulta se refiere a dos o mas atributos que tienen el mismo nombre, se debe **calificar** el nombre del atributo con el nombre de la relación a fin de evitar la ambigüedad.

**SELECT** Nombre, EMPLEADO.apellido1, dirección

**FROM** EMPLEADO,DEPARTAMENTO

**WHERE** DEPARTAMENTO .Nombre=’investigacion’ and DEPARTAMENTO.NumeroDpto=EMPLEADO.dno



//refiriéndose dos veces a la misma relación

**SELECT** E.Nombre, E.Apellido1, S.nombre, S.Apellido1

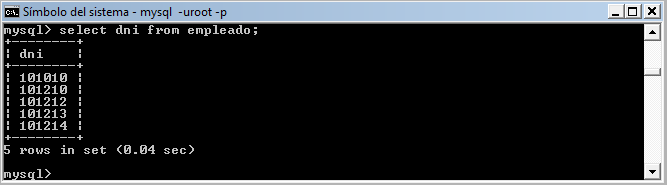
**FROM** EMPLEADO as E, EMPLEADO as S

**WHERE** E.SuperDni=S.Dni

* Clausula where no especificada y uso de arterisco

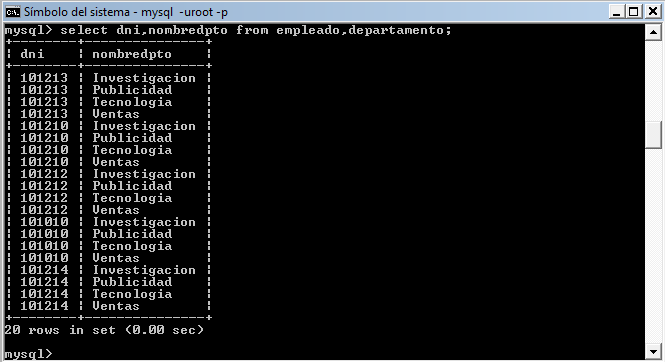
**SELECT** Dni

**FROM** EMPLEADO;



**SELECT** Dni, NombreDpto

**FROM** EMPLEADO, DEPARTAMENTO;



// Usando arterisco

**SELECT** \*

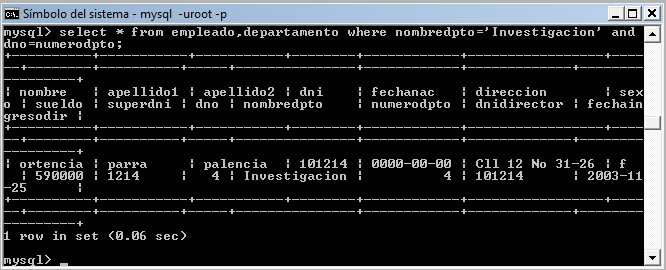
**FROM** EMPLEADO

**WHERE** Dno=5;

**SELECT** \*

**FROM** EMPLEADO,DEPARTAMENTO

**WHERE** NombreDpto=’investigacion’ and Dno=NumeroDpto;



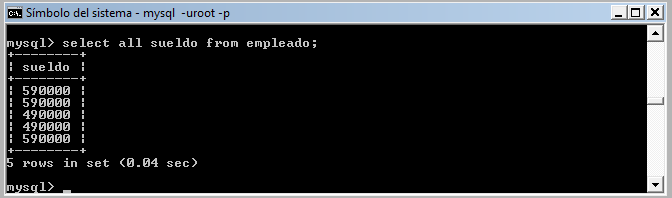
**SELECT** \*

**FROM** EMPLEADO, DEPARTAMENTO;

* Tablas como conjuntos en SQL

**SELECT ALL** Sueldo

**FROM** EMPLEADO;



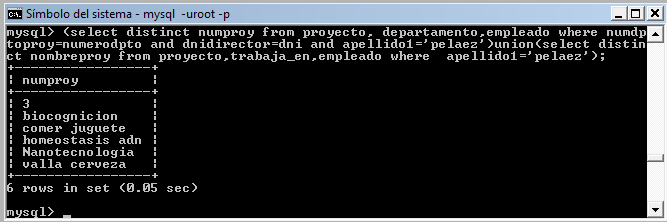
**SELECT DISTINC** Sueldo

**FROM** EMPLEADO;



// Utilizando operaciones del algebra relacional union de conjuntos **(UNION)** , Diferencia de conjuntos **(EXCEPT )**e intersección de conjuntos**(INTERSECT)**

(SELECT DISTINC NumProyecto FROM PROYECTO, DEPARTAMENTO,EMPLEADO where NumDptoProyecto=NumeroDpto and DniDirector=Dni and Apellido1=’Perez’ ) **UNION** (SELECT DISTINC NumProyecto FROM PROYECTO, TRABAJA\_EN, EMPLEADO WHERE NumProyecto= NumProy and DniEmpleado=Dni ADN Apellido1=’Pelaez’);

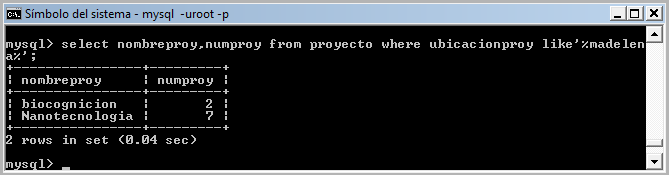


* Comparacion de subcadenas y operadores aritméticos

**SELECT** Nombreproy, numproy

**FROM** proyecto

**WHERE** ubicaionproy LIKE ’%Madelena%’



**SELECT \* FROM** EMPLEADO **WHERE** (Sueldo **BETWEEN** 30000 AND 4000) AND Dno=5;

La condición (Sueldo **BETWEEN** 30000 AND 4000) es equivalente ((Sueldo **>=**30000 )AND (Sueldo <= 40000))

* Ordenación del resultado de una consulta

**ORDER BY**

**SELECT** NombreDpto, Apellido1, Nombre, NombreProyecto

**FROM** DEPARTAMENTO, EMPLEADO, TRABAJA\_EN, PROYECTO

**WHERE** NumeroDpto= Dno and Dni= DniEmpleado and NumProy=NumProyecto

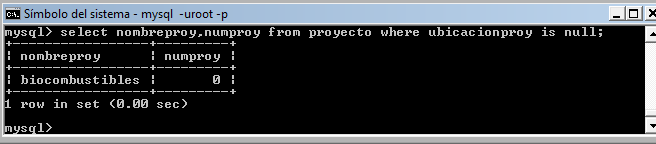
ORDER BY NombreDpto, Apellido1, Nombre;

* Comparaciones con valores NULL y lógica de tres valores
* *Valor desconocido:* Una persona en particular tiene una fecha de nacimiento, pero no la conocemos, por lo que la representamos con NULL en la base de datos;
* *Valor no disponible o no especificado:* Una persona tiene un teléfono en casa, pero no quiere que aparezca en el listado, por lo que se impide su visualización y se representa como NULL en la base de datos
* *Atributo no aplicable:* un atributo último grado seria NULL para una persona que no tiene licenciatura, algo que no es aplicable para esa persona.

**SELECT** Nombreproy, numproy

**FROM** PROYECTO

**WHERE** UBICACIÓN **IS NULL**;



* Consultas anidadas, tuplas y comparaciones conjunto / multiconjunto

**SELECT DISTINC** NumProyecto

**FROM** PROYECTO

**WHERE** NumProyecto **IN**(**SELECT** NumProyecto FROM PROYECTO,DEPARTAMENTO, EMPLEADO **WHERE** NumDptoProyecto = NumeroDpto and DniDirector=Dni and Spellido1=´Perez´) **OR**

NumProyecto **IN**(**SELECT** NumProy FROM TRABAJA\_EN, EMPLEADO **WHERE** DniEmpleado= Dni and Apellido1=’Pérez’);

**SELECT** DISTINC DniEmpleado

**FROM** TRABAJA\_EN

**WHERE** (NumProy,Horas)IN(SELECT NumProy,Horas FROM TRABAJA\_EN WHERE Dni=’12345678’);

**SELECT** Apellido1, Nombre **FROM** EMPLEADO **WHERE** Sueldo> **ALL** (SELECT FROM EMPLEADO WHERE Dno=5);

**SELECT** E.Nombre, E.Apellido1 FROM EMPLEADO AS E E.Dni **IN**(**SELECT** DniEmpleado FROM SUBORDINADO **WHERE** E.Nombre=NomSubordinado and E.sexo=sexo);

* Funciones EXISTS en SQL

La función **EXITS** de sql si el resultado de una consulta anidada correlacionada este vacío (no contiene tuplas) o no. ---valor booleano---

**NOT EXIST**

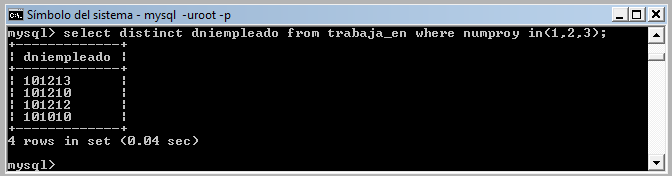
* Conjuntos explícitos y renombrado de atributos SQL

Donde el conjunto debe ir entre paréntesis

**SELECT DISTINCT** DniEmpleado

**FROM** TRABAJA\_EN

**WHERE** Numproy **IN** (1,2,3)



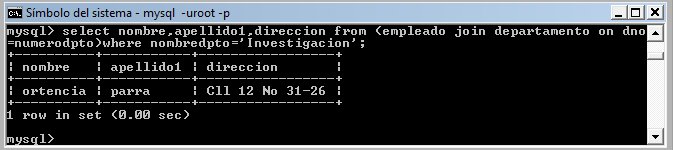
* Tablas concatenadas en SQL y concatenaciones exteriores

El concepto de tabla concatenada (o **relación concatenada**) se incorporo en SQL para poder especificar una tabla como resultado de una operación de concatenación en la clausula from de una consulta . Esta estructura es más fácil que mezclar todas las instrucciones de selección y concatenación en la clausula WHERE. Por ejemplo considere la consulta que recupera el nombre y la dirección de los empleados que trabajan para el departamento ‘Investigación’. Puede ser más fácil especificar primero la concatenación de las relaciones EMPLEADO DEPARTAMENTO, y después seleccionar las tuplas y los atributos deseados. Esto se puede escribir en SQL como:

**SELECT** nombre, apellido1, dirección

**FROM** (EMPLEADO JOIN DEPARTAMENTO ON Dno=NumeroDpto)

**WHERE** NombreDpto=’Investigación’.



// Con natural join no especifica condición de concatenación alguna

**SELECT** nombre, apellido1, dirección

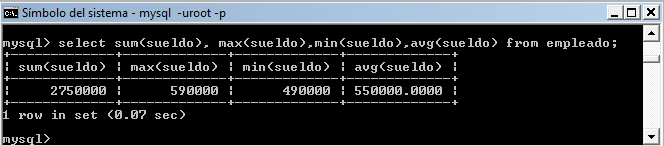
**FROM** (EMPLEADO NATURAL JOIN DEPARTAMENTO AS DEPT(NombreDpto, Dno, DniDirector, FechaDirector, Fechaindirector) ))

**WHERE** NombreDpto=’Investigación’.

* Funciones agregadas en SQL

**SELECT SUM** (Sueldo) , **MAX**(Sueldo), **MIN**(Sueldo), **AVG**(Sueldo)

**FROM** EMPLEADO;



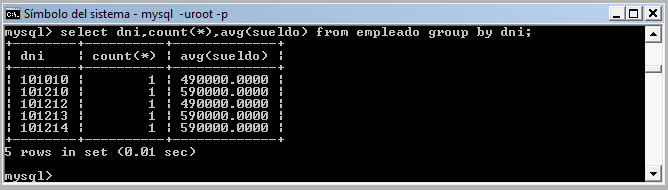
* Agrupamiento: Las clausulas **GROUP BY** y **HAVING**

**GROUP BY:** Especifica los atributos de agrupamiento, que también deben de aparecer en la clausula SELECT.

**SELECT** Dni, COUNT (\*), AVG (Sueldo)

**FROM** EMPLEADO

**GROUP BY** Dni;



**HAVING:** Proporciona una condición en el grupo de tuplas asociado a cada valor de los atributos de agrupamiento (se puede combinar con GROUP BY).

**SELECT** NUmproyecto, NombreProyecto, COUNT(\*)

**FROM** PROYECTO,TRABAJA\_EN

**WHERE** NumProyecto= NumProy

**GROUP BY** NumProyecto, NombreProyecto

**HAVING** COUNT (\*) >2;



* Explicación y resumen de las consultas de SQL

**SELECT** < lista de atributos y funciones>

**FROM** <lista de tablas>

[ **WHERE** <condición>

[ **GROUP** BY < atributos(s) de agrupamiento> ]

[**HAVING** < condición de agrupamiento>]

[**ORDER BY** < lista de atributos>]

* Sentencias INSERT, DELETE Y UPDATE de SQL

**INSERT:** Para añadir un tupla a una relación o tabla.

INSERT INTO PROYECTO values (‘Nanotecnologia’,7,’madelena’,2);

**DELETE:** Elimina tuplas de una relación. Incluye una clausula WHERE, para seleccionar las tuplas que se van a eliminar

**DELETE** FROM EMPLEADO

**WHERE** Apellido1=’Peláez’;

**UPDATE:** Se utiliza para modificar los valores de atributo de una o más tuplas. Una clausula SET especifican los atributos que se modificaran y sus nuevos valores.

**UPDATE** Proyecto

**SET** UbicacionProyecto=’Restrepo’, numdptoproy=1

**WHERE** Numproyect=1;

* Restricciones como aserciones y triggers

Se pueden especificar restricciones generales mediante aserciones declarativas utilizando la sentencia CREATE ASSERTION. A cada aserción se le asigna un nombre de restricción y se especifica a travez de una clausula parecida a la WHERE.

**CREATE ASSERTION** RES\_SUELDO

**CHECK** **(NOT EXISTS**(**SELECT** \* **FROM** EMPLEADO E, EMPLEADO M, DEPARTAMENTO D **WHERE**

E.Sueldo > M.Sueldo **AND** E.Dno= D.NumeroDpto **AND** D.Director=M.Dni));

**CREATE TRIGGER:** Especifica un evento (por ejemplo, una operacion de actualizacion de la base de datos ), una condición y una acción . La acción se ejecuta automáticamente si se satisface la condición cuando se produce el evento.

**VISTAS (TABLAS VIRTUALES) en SQL**

Una vista en terminología SQL es una tabla que deriva de otras tablas. Esas otras tablas pueden ser base o vistas definidas anteriormente. Una vista no existe necesariamente en un formato físico; está considerada como un tabla virtual, en oposición a las tablas base, cuyas tuplas están realmente almacenadas en la base de datos. Esto limita las posibles operaciones de actualización que pueden aplicarse a las vistas, pero no ofrecen limitación alguna al consultar una vista.

**CREATE VIEW** TRABAJA\_EN1 **AS SELECT** Nonmbre, apellido1, nombreproyecto, horas **FROM** EMPLEADO, PROYECTO, TRABAJA\_EN WHERE Dni= DniEmpleado AND NumProy=Numproyecto.

// borrado= drop view TRABAJA\_EN;

* El problema de implementar eficazmente una vista para consultas es complejo. Se han suregido dos metodologías principalmente . La denominada modificaion de consulta implica modificar la consulta de la vista para que sea una consulta de las tablas sabe subyacente. Por ejemplo

**SELECT** Nombre, Apellido1

**FROM** EMPLEADO, PROYECTO, TRABAJA\_EN

**WHERE** Dni=m DniEmpleado and Numproy= numproyecto

**And** nombreproyecto=’Nanotecnologia’;

UPDATE PROYECTO SET NombreProyecto=’Nanotecnologia’

WHERE NombreProyecto=’Nanotecnologia’